Implementation of NN for Edge AI

Author: Coby Cockrell
Date: 4/30/2024
Email: cockrellc2@vcu.edu

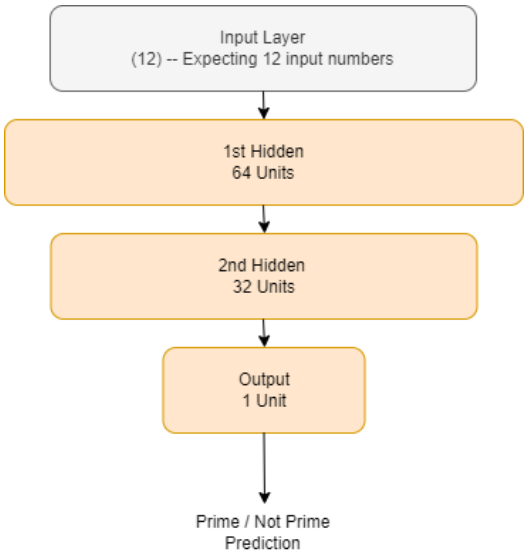
1. Abstract

The purpose of this document is to further investigate and analyze the implementation strategy utilizing Vivado/Vitis HLS for custom Neural Networks / SDeep Neural Networks (SDNN) architectures. Additionally, it should be remarked that not all AI models are capable of being fully 100% realized, as deeper architectures can pose additional challenges to the space of fabric available.

2. Objective

The example workflow utilized can be realized and adapted for creating other personalized and fully customized examples. Thus, it is essential to provide clear communication of the design steps, as such lets overview the following problem statement and architecture.

The simple classification chosen for this case is a prime number classification architecture, and as such the architecture will consist of Input layers, Dense layers, and Activation layers. The proposed initial architecture for testing can be seen in the following hierarchy :



3. **Initial Design**

Since the target HLS solution I've decided to go with is Vitis HLS the initial design language of choice is C++. Utilizing C++ allows Vitis HLS a smooth translation and gives many options for hardware customizations in later steps. For initial design, I have come up with a hierarchy of the C++ and header files that are needed to construct the MLP.

Bold = Made and Tested

Prime - Number MLP	headers/	src/	data/	weights/	testbenches/
	mlp.h	mlp.cpp	**train.txt**	weights.txt	mlp_Testbench.cpp
	activation.h	activation.cpp	test.txt		act_Testbench.cpp
	layer.h	**layer.cpp**			**layer_Testbench.cpp**
	utils.h	**utils.cpp**			**utils_Testbench.cpp**
		main.cpp			main_Testbench.cpp

For each major component (utils, activation, layer, and mlp) I plan on creating small additional testing scripts to verify each component. This is also a critical step for HLS, as included testbenches are a must. It is also important to keep in mind as this design progresses, the end implementation is an SoC, in which we are able to utilize the PS-PL partitioning to effectively speed-up our system. With this comes some file tracking, since the majority of the network is going to be implemented onto the PL the following files will undergo HLS transformation: mlp.h/mlp.c, activation.h/activation.c, and layer.cpp/layer.h.